

Establishing Latch Correspondence for Embedded Circuits of PowerPC[®] Microprocessors

H. Anand, J. Bhadra, A. Sen, M. S. Abadir, and K. G. Davis

Freescale Semiconductor Inc.

7700 West Parmer Lane

Austin, TX - 78727, USA

Contact email: Himyanshu.Anand@freescale.com

Abstract

We present a novel latch mapping methodology that judiciously leverages structural and functional analyses on digital sequential circuits. We make use of functional design constraints in a novel way to get latch correspondence information. For scanable latches we use a technique based on scan chain analysis to obtain latch correspondences. We also provide an effective heuristic for finding latch correspondences for latches (potentially non-scanable) in complex state machines having cyclic dependencies. Our methodology not only answers latch correspondence, but also provides polarity of the correspondence. This is a major advantage over earlier latch mapping algorithms. Experimental results obtained on embedded circuits from live PowerPC[®]¹ design projects have shown that our technique fares better than a leading vendor tool in mapping latches – in both quantitative (more latches mapped) and performance (time/memory used) aspects.

1 Introduction

In high performance design projects such as the PowerPC line of processors, for timing, power and other performance issues, typically one does not employ synthesis techniques. Therefore, for all performance critical sub-circuits, a manually written Register Transfer Level (RTL) design is treated as the specification while a manually designed transistor circuit level (TCL) model serves as the implementation. Unlike ASIC flows, there is no synthesis involved in the translation from RTL to TCL. Therefore, equivalence checking is an important facet of implementation verification.

Two models of the same design can be checked for equivalence using either sequential [1] [6] or combinational equivalence checkers [9] [7]. Sequential equivalence checkers verify that the outputs of two models have the same values for all reachable states in

the models. Combinational equivalence checkers verify that corresponding combinational blocks of two models are equivalent. Two steps are necessary in combinational equivalence checking: (a) identifying the correspondence between the latches of the two design descriptions such that the circuit models can be decomposed into corresponding combinational blocks and (b) verifying that the corresponding combinational blocks are functionally equivalent.

In our flow, combinational equivalence checking is used to guarantee equivalence between the two manually generated circuit models.

Latch correspondence between the two models is often not obvious. For complex state machines like high performance pipelines, bus and other control having latches with sequential cyclic dependencies, the notion of manual generation of latch correspondence is non-trivial. This is firstly because of the complexity of the circuits. Secondly and more importantly, the specification of these complex circuits might be too abstract to contain all the latches present in the implementation. Therefore, manually providing latch correspondence is complex, time consuming and error prone. This complicates debugging and increases equivalence checking cycle time and time to market. Automating the process of identifying latch correspondence makes the turn around time for combinational equivalence checking faster and more accurate.

Recently there has been a lot of interest in combinational equivalence checking because of their efficiency, and as a result considerable research has been done on techniques for computing latch correspondence [2] [4] [5] [7] [8] [13]. There are techniques that *automatically* derive latch correspondences using heuristics based on signal names and structural analysis of the models [4] [5], iterative refinement of the correspondences [7] [8], a combination of both of the above methods [2], random simulation [4], or methods based on functional comparison [7].

The latch correspondence approach presented by Cho and Pixley [4] uses a methodology based on structural and simulation analysis. Their structural analysis tech-

¹PowerPC name is trademark of IBM Corp. and used under license.

nique suggests that latches correspond to each other if their fanin (fanout) cones are fed by corresponding latches. Their methodology uses *random* simulation of the models to find latch correspondences by comparing the values that are stored in the latches on successive cycles during the simulation. Their technique fails to identify that random simulation can violate the functional constraints for the two models being compared. That is why their approach may calculate incorrect correspondences because the random simulation may contain stimuli under which the models are not equivalent. Our methodology uses *constrained simulation* of the models by modeling of the environmental assumptions as constraints on the primary inputs. Moreover, simple fanin/fanout analysis [4] will fail to find latch correspondence in cyclic structures formed by sequential feedback. The algorithms presented by van Eijk and Jess [8] and Burch and Singhal [7] both use functional analysis techniques based on fixed-point iterations to refine the set of all latches into a correct and complete correspondence. However, functional analysis techniques suffer from the well-known *state explosion problem*. ATPG techniques have also been used for latch correspondence [5] but the technique is not guaranteed to provide correct or complete set of maps. A technique arrived at by combining a set of previously known techniques was presented by Ng et al [2].

Our methodology presents a set of mutually cooperating *filters* for calculating latch correspondences along with polarity information. We propose a methodology that consists of four filters – a simplistic structural analysis [4], *constrained simulation*, *scan chain analysis* and *distance metric analysis*. The overall tool inputs a pair of models, functional constraints and correspondence between the models’ primary inputs and outputs. At each iteration, each filter works its inputs and a (potentially empty) set of partial latch correspondence obtained from the previous filter/iteration and calculates new latch correspondences and forwards that information onto the next filter/iteration. At any iteration, if after passing through all the filters, if the set of latch correspondence remains the same, then we terminate the method and output the current latch correspondence and polarity information. Since the total number of latches is finite, the procedure always terminates. To the best of our knowledge ours is the first reported technique that provides latch correspondence along with true/inverted polarity of the correspondence – a piece of information that is invaluable in combinational equivalence checking.

This paper makes the following contributions:

- We use functional constraints to guide simulation
- We use a novel scan chain analysis technique
- We provide a heuristic that calculates latch correspondence for latches having complex feedback

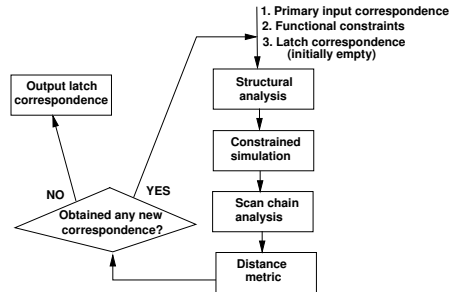


Figure 1: Filters in Latch Correspondence

dependencies

- We validate the effectiveness of our technique on industrial scale embedded circuits

2 Simulation Analysis

Structural analysis coupled with levelization (level of a latch is maximum number of latches in a path from any primary input to that latch without crossing a sequential cycle) is an effective first filter. We augment it using algorithms from simulation analysis and a heuristic using distance metrics. Simulation also provides a fast way to find true/inverted polarity.

In simulation analysis, we generate a test bench environment, where the two models are simulated in parallel with a set of input vectors and latch values are checked for true or inverted correspondence. After the simulation is performed we put the latches into bins, based on their simulation signatures. If a bin has only two latches, one from each model, then these latches are candidates for correspondence. Simulation analysis results are checked against the correspondence found using structural analysis. If the structural analysis also finds the same correspondence as simulation analysis, then simulation analysis determines whether it is a true or inverted polarity.

Careful considerations must be given to the fact that if the constraints provided during simulation analysis are incorrect then simulated states may be unreachable in reality and the simulation analysis data has to be discarded. However, using correct constraints with reset sequences can guarantee that this is never the case. Also, since we simulate for a finite length of input vectors, the vector set may not be large enough to differentiate between two non-corresponding latches. However, in the absence of a bug, constrained simulation with reset sequence would never give false positives.

In our experiments in the absence of reset sequences, initializing all latches in the two models to ‘X’ generated correct results. Initializing all latches to all ‘0’s or all ‘1’s inherently assumes all true or all inverted latch correspondences that in turn may initialize the models

in unreachable/non-equivalent states. A typical design has a mix of true and inverted latch correspondences and so starting the simulation from an all ‘X’ states is necessary in the absence of reset sequences.

3 Constraining Models

We model the environment of the two designs under consideration using *environmental design constraints* [10] [11] [12], often referred to simply as *functional constraints*. In our experience, constraining the models’ primary inputs has been satisfactory for achieving latch correspondence.

We use functional constraints in our latch correspondence methodology in order to increase its effectiveness and to avoid false positives while finding corresponding latches. Let us assume that the *binary state space* of the design under consideration is S (Figure 2), where each element of S is a *state* (A, B, \dots, J, K etc.) such that each latch gets a unique value of ‘0’ or ‘1’. Naturally, if there are n latches in the design, $|S| = 2^n$. Also, let C represent the functional constraint under which the model is to be instantiated in the microprocessor design. The effect of C can be looked upon as *restricting* the state space of the design. Let S_C be the constrained state space that is actually of interest for functional verification. We have $S_C \subseteq S$. Also of interest is the *complete state space*, U , where each latch can have values from the set $\{0, 1, X\}$. Naturally, $S \subseteq U$.

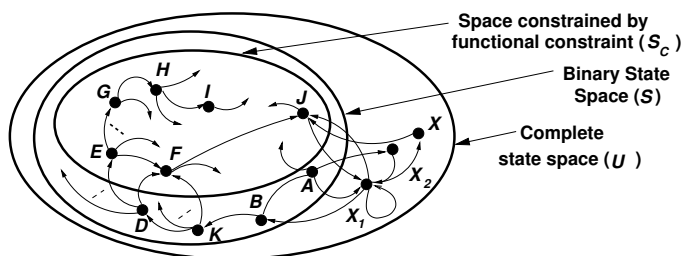


Figure 2: Functional Constraint and Latch Correspondence

We begin our simulation from a special state (say, $X_1 \in U - S$) where all latches have the value, ‘X’. States in the space $U - S$ represent *all* states that have *at least* one latch that is in the ‘X’ state. Also, states in space $U - S$ could have transitions to and from states in S_C , S and $U - S$. Since we initialize all our latches to ‘X’ values, our simulation always begins at the space $U - S$, and proceeds along transitions that emanate from the space $U - S$ going to states that are members of $U - S$ or that of S .

Now consider the fact that the equivalence between the two models holds *only* when the two designs are constrained by the functional constraint C . In case the simulation trace leaves the space constrained by C , i.e., in

state space $U - S_C$, then there can be no claim as to the equivalence between the two models. This in turn implies that two latches that are equivalent in state space S_C could behave completely differently if the simulation follows a path such as $X_1 \rightarrow B \rightarrow K \rightarrow D \rightarrow \dots$ (where, $X_1 \in U - S$) because the entire path belongs to $U - S_C$. If the design is not constrained using the correct functional constraint C , then it leads to problems - Case 1: if the simulation is not constrained within S_C , then two latches that are supposed to be equivalent within S_C might behave differently resulting into different simulation signatures and so we might not be able to find latch correspondence. Case 2: if we simulate the design in the space $U - S_C$, then the simulation might just be such that two non-equivalent latches happen to exhibit the exact same simulation signature and are wrongly marked as corresponding latches. We prevent the above two situations by (a) using functional constraints, and (b) by using structural analysis data for getting the final correspondence between the latches. However, to completely eliminate the above situations, reset sequences have to be applied for taking the designs from $U - S_C$, to S_C . Functional simulation signatures that take the designs out of S_C cannot be relied upon for latch correspondence due to problems stated above.

4 Scan Chain Analysis

In our design house most of the latches in high performance designs have full-scan features. Analysis of scan chains greatly improves our results by leveraging both structural and simulation analysis. Figure 3 shows a typical scan chain set up.

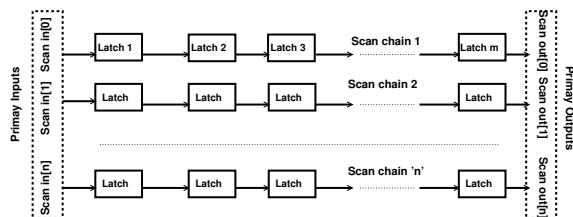


Figure 3: A typical scan chain set up

Scan chain analysis can be directly performed in circuits that do not have any memory cores. In case the design has a memory core in it, the analysis is described as follows. We design memory sub-circuits such that the memory core is sandwiched between two sets of latches along with surrounding logic. We illustrate this in Figure 4 where solid arrows represent scan chains, and dashed arrows represent logical dependencies.

Functional analysis techniques [8] [3] for latch correspondence do not scale for memory designs, because they suffer from *state explosion problem*. In cases where there are *output latches* the latch values read out of the

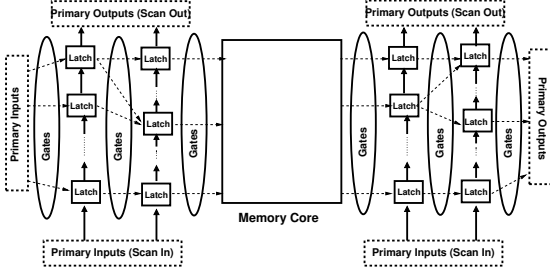


Figure 4: A Typical Memory Design

memory core depend on all the bits in the addressable space of the memory core. Let us consider a memory with 8K words where each word is 8 bits wide. Since the read port of the memory would be 8 bits wide there would be 8 output latches capturing the read data. Each of these latches would depend on at least 8K bits that feed into them.

Similarly, generating *random patterns* at inputs and comparing values of memory cores does not scale well for memory bit correspondence. Hence, we black-box all our memory cores such that all the output pins of the core are nondeterministic. Once the memory core has been black-boxed, we cannot use functional patterns for determining latch correspondence as output of the core is functionally independent of its inputs. Scan chain analysis plays a vital role in both avoiding state space explosion and overcoming the limitations imposed by black-boxing of the memory core.

Simple structural analysis of scan chains may not be able to find all latch correspondences as it does not differentiate between scan paths and functional paths. Additional information such as naming conventions can however be used to enhance the analysis and find latch correspondences on the scan chains.

Scan chains also improve the simulation analysis. We *constrain* the models such that during simulation only scan path is sensitized, hence comparison of simulation signatures leads to latch correspondence. This assumes that the scan paths do not contain any bug and are unique. In our experience bugs are typically less in scan chain paths compared to functional paths, as scan chains generally contain little logic, if any.

Example: Let us assume that the only scan chain in the models is scan chain 1 (Figure 3) with $m = 4$. Let us set up the scan mode such that ‘1’s and ‘0’s are scanned in through the primary input pin ScanIn[0]. Initially, all four latches have the value ‘X’. When the scan mode is on, the circuit starts simulating from the special state $X \in (U - S)$ that had all latches in the ‘X’ state. After a ‘1’ is scanned in through primary input ScanIn[0], the value in Latch1 is ‘1’ and all others are ‘X’s (state ‘1XXX’). After scanning in three more inputs (say, 110; 0 scanned in last), we have state $s = ‘0111’$. Clearly the first four states ($X = ‘XXXX’$,

$s_1 = ‘1XXX’$, $s_2 = ‘11XX’$, $s_3 = ‘111X’$) implied that the state machine is in the space $U - S$, but the last state $s \in S$. Using only functional analysis would require us to discard all the five states (of the simulation trace $X \rightarrow s_1 \rightarrow s_2 \rightarrow s_3 \rightarrow s$) that were reached during the scan mode simulations because $X, s_1, s_2, s_3 \notin S_C$ and s also may not be in S_C . However, since the scan chains are present in both the models, we can easily compare the values in the corresponding scan chains and obtain latch correspondence even when the simulation trace did not conform to S_C . Thus we can diminish the damage done by black-boxing the memory core by *exploiting* scan chains during simulation.

In the presence of multiple scan chains, we have to guarantee two criteria in order to do the same analysis. First, each latch should have a unique simulation signature at the end of the scan mode simulation. Second, in order to determine the polarity of the latch correspondence, no two latches in the same model can have the exact inverted signature with respect to each other (i.e., there can not be two latches in the same model with simulation signatures 110001 and 001110). These two conditions ensure that a given latch in one model can correspond (true or inverted) to exactly one latch in the other model.

Input vectors to meet the above conditions can be easily generated if the scan chains do not contain any logic. If, however, in the event of a scan chain having logic inside it, the problem of generating unique patterns on latches is an ATPG problem. In such a case, random scan-in might be more effective. Also if two models do not have identical state encodings, then functional techniques might provide some correspondences if they exist.

5 Mapping Latches with Cyclic Structures

It has been shown earlier that structural fanin/fanout analysis is a very efficient and effective technique for finding latch correspondences in industrial designs [4]. Structural techniques are especially effective when the designs are similar to each other. In general, these techniques assume that there are no cycles in the designs. In this section, we extend structural fanin/fanout analysis techniques to structurally similar designs that contain sequential cycles but no scan chains. These kind of circuits are frequently encountered in high performance pipelines, protocols and other control circuitry.

In earlier work [4], latches in both designs were partitioned into levels; then, starting from the first level (primary inputs), fanin/fanout information of latches is used to compute the latch correspondences between designs for every level. In other words, when computing the latch correspondences for level i , the latch corre-

spondences for levels up-to and including level $(i - 1)$ is already known. Based on levelization, correspondence criterion in earlier work is as follows: Two latches n_1 and n_2 both at level i *correspond* to each other if *for every* fanin latch x at level $(i - 1)$ of n_1 there is a fanin latch y at level $(i - 1)$ of n_2 such that x corresponds to y and vice versa, and there is no other latch that corresponds to n_1 or n_2 . We use *fanin* of a latch to denote latches and primary inputs in cone of logic of a latch. *Fanout* of a latch is defined similarly.

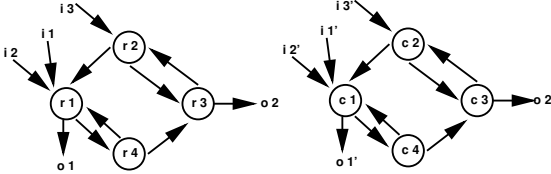


Figure 5: Cycle of length four

The criterion for latch correspondence defined above is quite restrictive and cannot handle cyclic structures. The restriction manifests itself in the expectation that all the fanins of a latch must have correspondence, which is violated for latches in a cycle because every latch in the cycle has in its fanin another latch in the same cycle that has no correspondence yet.

What if we consider a cycle (all latches in a cycle) as a single latch and assign the cycle a level such that it is 1 greater than maximum of all the fanins of the latches in the cycle? In this case, even after assuming that all the fanin correspondences of a cycle have been resolved, all the latches in the cycle will have the same fanin, hence the criterion above will fail to find correspondences again. Specifically, when applied to the example in Figure 5, the criterion will find no correspondences. However, we should be able to find the correspondences $r1 = c1$, $r2 = c2$, $r3 = c3$, and $r4 = c4$. This is because the fanin and fanout information, $i1 = i1'$, $i2 = i2'$, $i3 = i3'$, $o1 = o1'$, $o2 = o2'$ is sufficient to resolve correspondences for the cycle in the figure. Next we will formalize our intuitions and obtain a heuristic that can find correspondences efficiently.

5.1 Distance Metric

We have developed a metric that allows us to find correspondences in cycles. We use a tuple *distance* to find latch correspondence between two latches r_1 and c_1 . Formally, $distance(r_1, c_1) = \langle l, I, CL, CFI, FI, CFO, FO \rangle$ consists of the following integer components

1. A number $l \in \{0, 1\}$, which is 1 if both the latches belong to the same level, otherwise 0.
2. A number $I \in \{0, 1\}$, which is 1 if both the latches belong to a cycle or neither belong to a cycle, otherwise 0.

3. A number $CL = norm(|CL_{r_1} - CL_{c_1}|)$, which determines whether both latches belong to cycles with similar length and where CL_{r_1} is the length of the cycle that contains r_1 .

4. A number CFI , which is the number of corresponding fanins of r_1 and c_1 .

5. A number $FI = norm(|FI_{r_1} - FI_{c_1}|)$, which determines whether both latches have similar number of fanins and where FI_{r_1} is the number of fanins of r_1 .

6. A number CFO , which is the number of corresponding fanouts of r_1 and c_1 .

7. A number $FO = norm(|FO_{r_1} - FO_{c_1}|)$, which determines whether both latches have similar number of fanouts and where FO_{r_1} is the number of fanouts of r_1 .

We use a normalization function $norm(a) = (-1)a$ to normalize the numbers such that the higher the number given to the function the lower the number it returns.

Given two designs M_r and M_c , we define the set of all *distance*'s as $F = \{distance(r_i, c_j) : i \in \text{latches in } M_r, j \in \text{latches in } M_c\}$. We use lexicographical comparison (dictionary ordering) to order two *distance*'s, (a_1, a_2, \dots, a_7) and (b_1, b_2, \dots, b_7) from F . Given the usual ordering on numbers $<$, the distance ordering $<^d$ is then

$$(a_1, a_2, \dots, a_7) <^d (b_1, b_2, \dots, b_7) \iff (\exists m > 0) (\forall i < m) (a_i = b_i) \wedge (a_m < b_m)$$

That is, if one of the terms $a_m <_m b_m$ and all the preceding terms are equal, we have $<^d$. We use $(a_1, a_2, \dots, a_7) \leq^d (b_1, b_2, \dots, b_7)$ to denote that $(a_1, a_2, \dots, a_7) <^d (b_1, b_2, \dots, b_7)$ or $(a_1, a_2, \dots, a_7) = (b_1, b_2, \dots, b_7)$. The above selection of tuples gives higher weight-age to metrics that appear early in the tuple. Thus, $weight_{a_1} > weight_{a_2}$.

Figure 6 illustrates our distance metric algorithm (DistMet). We compute correspondences starting from the first level (Step 1). We use a distance matrix G with n rows and m columns, where n and m are the number of latches from circuits M_r and M_c at the current level, respectively. We use J to keep the sorted list of pairs from G . We use Y to keep the set of largest pairs from J and H to keep the correspondences that are found so far. The algorithm has two while loops. The outer while loop (Steps 4-17) is executed as long as a correspondence has been found in a previous iteration of the while loop and there are still correspondences that need to be found. In the outer while loop we compute the *distance* pairs in G and sort them in J (Steps 6-7). The inner while loop (Steps 8-16) iterates over the sorted pairs list J and decreases J 's size after each iteration. During each iteration of the inner while loop, we compute Y (Step 9) and remove all rows and columns in Y from J , that is, if $J = \{(r_1, c_2), (r_2, c_3), (r_3, c_3)\}$ and $Y = \{(r_2, c_3)\}$ then $J = \{(r_1, c_2)\}$ (Step 10). Hence, we can focus on the "next" largest set of pairs in J

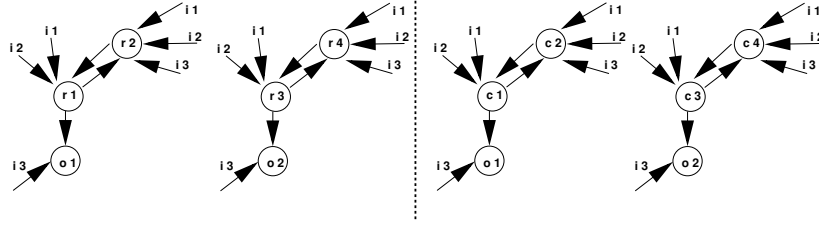


Figure 7: Two Cycles of the same length

Algorithm DistMet

Input: Circuits M_r and M_c

Output: Set of latch correspondences

0. let $H := \emptyset$;
1. for each level l
2. let G be matrix of *distance* tuples initially empty;
3. let $Y := \emptyset$;
4. do
5. found := false;
6. compute *distance* for each (row, column) pair in G ;
7. let J be the list of sorted pairs in G according to $<^d$;
8. do
9. $Y :=$ set of all largest pairs in J ;
10. remove all rows r_1 and columns c_1 in Y from J ;
11. remove all rows r_1 and columns c_1 that exist more than once in Y from Y ;
12. if $Y \neq \emptyset$ then
13. found := true;
14. remove all rows r_1 and columns c_1 in Y from G ;
15. let $H := H \cup Y$;
16. endif;
17. while ($J \neq \emptyset$) do
18. while (found & $G \neq \emptyset$)
19. endwhile;
20. endwhile;
21. return H ;

Figure 6: Distance Metric Algorithm (DistMet)

at the next iteration of the inner while loop. At Step 11, we remove all the pairs in Y that are intersecting, that is pairs that have a row or column in common, e.g., $Y = \{(r_1, c_2), (r_1, c_3), (r_2, c_4)\}$, r_1 exists in both $(r_1, c_2), (r_1, c_3)$ so $Y = \{(r_2, c_4)\}$. At this point, if $Y = \emptyset$ then all the pairs in Y were intersecting therefore we can not find correspondence for the “current” largest set of pairs and we move to the next iteration of the inner while loop. However, if $Y \neq \emptyset$ (Step 12) then we found a correspondence (Step 13), and we remove all rows and columns in G that contain elements from Y (Step 14) so that each latch has a single correspondence. Finally we put the correspondence in H (Step 15).

5.2 Illustrative Example

In Figure 7, we consider a single level. Latches $\{r_1, r_2, r_3, r_4, o_1, o_2\}$ belong to circuit M_r and latches $\{c_1, c_2, c_3, c_4, o_1, o_2\}$ belong to circuit M_c . Also, the fanins i_1, i_2 and i_3 and the fanouts o_1 and o_2 correspond in both circuits. Note that we use o_1 and o_2 to denote both the fanouts and the latches. For simplicity, we will only show the matrix for correspondence between the latches in the cycles.

	c_1	c_2	c_3	c_4
r_1	(2, 0, 1, 0)	(2, -1, 0, -1)	(2, 0, 0, 0)	(2, -1, 0, -1)
r_2	(2, -1, 0, -1)	(3, 0, 0, 0)	(2, -1, 0, -1)	(3, 0, 0, 0)
r_3	(2, 0, 0, 0)	(2, -1, 0, -1)	(2, 0, 1, 0)	(2, -1, 0, -1)
r_4	(2, -1, 0, -1)	(3, 0, 0, 0)	(2, -1, 0, -1)	(3, 0, 0, 0)

	c_2	c_4
r_2	(4, 0, 1, 0)	(3, 0, 0, 0)
r_4	(3, 0, 0, 0)	(4, 0, 1, 0)

Table 1: Matrix G for iterations of DistMet for Figure 7

We first obtain the the upper table in Table 1. During the first iteration of the inner while loop we have $Y = \{(r_2, c_2), (r_2, c_4), (r_4, c_2), (r_4, c_4)\}$, hence all pairs are intersecting and Y is set to \emptyset at Step 11. However, J now contains the “next” largest pairs at the beginning, that is Y becomes $\{(r_1, c_1), (r_3, c_3)\}$. Clearly, none of the entries in Y are intersecting so we can add $\{(r_1, c_1), (r_3, c_3)\}$ to H . At this step $J = \emptyset$ hence, we recompute *distance* for the remaining entries in G and obtain the lower table in Table 1. $Y = \{(r_2, c_2), (r_4, c_4)\}$ and none of the pairs are intersecting so G becomes empty and $H = \{(r_1, c_1), (r_3, c_3), (r_2, c_2), (r_4, c_4)\}$.

6 Experimental Results

We have developed a tool, called ASM that finds latch correspondences of two circuit models. Most example circuits are in-house proprietary circuits that are used in PowerPC line of microprocessors. We compared our methodology with a commercial vendor tool, that finds latch correspondences automatically. All the experiments were run on 440 MHz, SunOS 5.8 Sparc machines

Table 2: Structural Analysis Results on Embedded Circuits

Circuit	ASM			Vendor			# Latches		% Mapping	
	Latches	T(sec)	M(MB)	Latches	T(sec)	M(MB)	Spec	Impl	ASM	Vendor
Ckt 1	1346	23	37	394	254	52	1346	1490	100	18.9
Ckt 2	164	13	35	164	4	50	164	164	100	100
Ckt 3	638	29	40	484	103	69	1018	1018	62.7	47.5
Ckt 4	81	216	392	267	164	88	368	322	25.1	82.9
Ckt 5	332	34	65	332	10	52	684	332	100	100
Ckt 6	1374	30	48	332	1082	52	1374	1374	100	24.2

with 1GB RAM. In the tables, we use ‘T’ to denote time and ‘M’ to denote memory usage. The percentage of the correspondence was calculated as the ratio of correspondences found by the tools to correct correspondences.

ASM was able to find all of the correspondences in circuits Ckt 1, Ckt 2, Ckt 5 and Ckt 6 by structural analysis alone as evident from Table 2. The latches in these circuits were connected in unique scan chains and structurally the two models of the designs were similar to each other.

Table 3: Results for constrained scan chain simulation

Circuit	ASM		# Latches	
	Latches	T(sec)	Spec	Impl
Ckt 1	1346	219.86	1346	1490
Ckt 2	164	30.16	164	164
Ckt 3	1018	142.35	1018	1018
Ckt 4	272	153.39	368	322
Ckt 5	332	74.71	684	332
Ckt 6	1374	169.69	1374	1374

Table 3 shows the results for doing structural and constrained simulation analysis for 1000 input vectors in *scan mode*. The really interesting cases are Ckt 3 and Ckt 4. In Ckt 3, even though the scan chains were present, pure structural analysis could not differentiate between all the latches. This is due to the fact that we did not do name matching and port matching. Also, if latch dependencies are not unique, then structural analysis cannot find all correspondences. However, when Ckt 3 was constrained to scan mode during simulation, we were able to find all correspondences, as the scan paths were unique. For Ckt 4, simulation did not yield all correspondences, as not all latches had a corresponding latch in the other model. However, the latches that were scanable were identified by constrained simulation.

Ckt 4, was also considerably more complex than other designs that gets reflected in the increased simulation time even though the latches are not that many as compared to other circuits. The reason for higher memory consumption is that the current implementation of ASM builds expression trees, that can be avoided during structural analysis.

Results on the distance metrics applied to circuits with cycles with varying degree of complexity and struc-

tures are given in Table 4. The circuits were designed to have different cyclic structures with no scan chains. In Cyc 1, both the cycles were at the same level. In Cyc 2, 10 cycles were at the first level after which all the cycles were at different levels. In Cyc 3, there were 20 levels with 3 cycles at each level, with , primary inputs feeding the cycles with lowest level. Cyc 6 had at the lowest block level a latch feeding a 3 latch cycle which then fed into 6 other latches forming a chain. This block was then replicated with some common inputs and one set of different inputs. Different inputs helped in identification of the latches driving the different cycles at the same level. Then, simple traversal of all levels in the chains resolved remaining correspondences. Cyc 7 is a relatively larger block where at the lowest level it had one cycle feeding another cycle. Cyc 8 had two latches in each cycle that had same fanin/fanout, hence were not differentiable even by distance metric. Cyc 9 was the most interesting case, that at the lowest level had 3 latches that formed a strongly connected component and were also part of latch chains. In addition there was latch that was not part of the cycle at all.

Our experimental results confirm that there are circuits that cannot be resolved by distance metric. However, as long as there are unique fanins and fanouts for latches, they can be resolved even if they are in a cycle. If circuits with different state encodings are given as inputs, then there does not exist latch correspondence for every latch. In such a case distance metric will find the closest latches as defined by the metric. Time and memory for processing larger designs did not increase dramatically, as we focused on creating different cyclic structures than putting lot of gates in the larger designs. The vendor tool was not able to find all the maps in Cyc 7, Cyc 8 and Cyc 9. In Cyc 7 and Cyc 8, vendor tool failed to identify any of the latches in the cycles, also it failed to identify some latches not in the cycle. In Cyc 9, the latches that vendor failed to resolve were at the end of chains. We do not know the internals of vendor tool, but the failure to identify the latches could be due to the fact that they may not be fully exploiting structural similarities.

Table 4: Experiments on Distance Metric

Ckt	PIs	POs	Latches	Cycles	Cycle Metric			Vendor		
					Maps	T(sec)	M(MB)	Maps	T(sec)	M(MB)
Cyc 2	30	10	60	20	60	0.19	11	60	0.19	39.99
Cyc 3	21	10	60	20	60	0.15	12	60	0.23	39.99
Cyc 6	61	30	180	60	180	0.20	12	180	0.22	39.97
Cyc 7	61	30	240	60	240	0.15	12	33	0.2	40.03
Cyc 8	61	30	360	60	240	0.25	12	33	0.21	40.03
Cyc 9	41	30	600	60	600	0.24	12	420	0.46	40.12

7 Advantages and Limitations

The main advantage of our techniques is that they are mutually complimentary and when used as filters, then after a couple of iterations most of the latch correspondence are known. The techniques used in conjunction with each other have proved to be more effective in our design style (partial scan, LSSD). Our techniques will be able to handle branching scan chains if they branch identically in two models. Also, our techniques are not dependent on naming conventions or any assumptions on rigid circuit structures.

There can be pathological cases under which the tool is misled to produce wrong latch correspondences when there are bugs in the models. This is true about *all other techniques* in the general area. In our technique impact of structural bugs (wrong connection) is more severe as compared to functional bugs (wrong logic), as apart from constrained simulation, other filters are not dependent on the functionality of circuits. Since we are interested in finding out latch correspondence, and not proving latch equivalence in the two models, the problem of finding out errors in the design is out of the scope of this work. Such bugs will be detected by equivalence checking tools when provided with incorrect latch correspondences.

8 Conclusions

Our methodology combines structural, simulation, scan chain and distance metric analysis techniques for finding latch correspondences in designs that cannot be handled by functional approaches or have sequential cycles. Our experiments validate the effectiveness of our methodology on high performance custom circuits. Our approach is orthogonal to other analysis techniques, hence can be used in conjunction with them.

References

[1] C. A. J. van Eijk, "Sequential Equivalence Checking without State Space Traversal", Proc. of Design Automation and Test in Europe, 1998

[2] K. Ng, M. R. Prasad, R. Mukherjee, J. Jain "Solving the Latch Mapping Problem in an Industrial Setting", Proc. of Design Automation Conference, 2003

[3] D. Brand "Verification of Large Synthesized Designs", Proc. of Intl. Conference of Computer Aided Design, 1993

[4] H. Cho, C. Pixley "Apparatus and Method for Deriving Correspondence between storage elements of a First Circuit Model and storage Elements of a Second Circuit Model", United States Patent number 5,638,381, dated 1997

[5] D. Anastasakis, R. Damiano, H.-K. T Ma, T. Stanion "A Practical and Efficient Method for Compare-Point Matching", Proc. of Design Automation Conference, 2002

[6] C. Pixley "A Theory and Implementation of Sequential Hardware Equivalence", IEEE Transactions on CAD, 1992

[7] J. R. Burch, V. Singhal "Robust Latch Mapping for Combinational Equivalence Checking", Proc. of Intl. Conference of ComputerAided Design, 1998

[8] C. A. J. van Eijk, J. A. G. Jess "Detection of Equivalent State Variables in Finite State Machine Verification", Proc. of Intl. Workshop of Logic Synthesis, 1995

[9] N. Krishnamurthy, A. K. Martin, M. S. Abadir J. A. Abraham "Validating PowerPC microprocessor custom memories", IEEE Design & Test of Computers, 17, 4, Oct-Dec, 2000

[10] M. Kaufmann, A. K. Martin, C. Pixley "Design Constraints in Symbolic Model Checking", Proc. of CAV, 1998.

[11] J. Bhadra, N. Krishnamurthy "Automatic Generation of Design Constraints in Verifying High Performance Embedded Dynamic Circuits", Proc. of ITC, 2002.

[12] J. Bhadra, N. Krishnamurthy, and M. S. Abadir "Enhanced Equivalence Checking: Toward a Solidarity of Functional Verification and Manufacturing Test Generation", IEEE Design and Test, November/December, 2004.

[13] J. Mohnke, P. Molitor, and S. Malik "Establishing Latch Correspondence for Sequential Circuits using Distinguishing Signatures", Integration, VLSI Journal, 1999.